Microtec
Research Inc.

MIGRATION MANUAL
Number 100671-002

Microtec Research

MCC68K ANSI C Compiler

Release 4.x

October 1, 1991

TRADEMARKS

Apollo™, DOMAIN/IX™, and AEGIS™ are trademarks of Apollo Computer, Inc.

High C® is a registered trademark of Metaware, Inc.

HP™ and HP-UX™ are trademarks of Hewlett-Packard Company.

IBM® is a registered trademark of International Business Machines, Inc.

ICE® is a registered trademark of Intel Corporation.

Microtec® and Paragon® are registered trademarks of Microtec Research, Inc.

MRI ASM™, MRI C™, MRI FORTRAN™, MRI Pascal™, and MRI XRAY™ are trademarks of Microtec Research, Inc.

MS-DOS™ is a trademark of Microsoft Corporation.

Sun™ is a trademark of Sun Microsystems, Inc.

Tektronix® is a registered trademark of Tektronix, Inc.

UNIX® is a registered trademark of AT&T.

VAX™, VMS™, and ULTRIX™ are trademarks of Digital Equipment Corporation.

XRAY68K™, XRAY86™, XRAY180™, XRAYZ80™, XRAY29K™, XRAYG32™, XRAY51™, XRAYTX™, XRAY Debugger™, XRAY In-Circuit Debugger™, and XRAY In-Circuit Debugger Monitor™ are trademarks of Microtec Research, Inc.

Zilog® is a registered trademark of Zilog Corporation.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 of DOD Regulation 27-401, or comparable clause, as such may be amended or renumbered from time to time.

MICROTEC RESEARCH, INCORPORATED
2350 MISSION COLLEGE BLVD.
SANTA CLARA, CA 95054

Table of Contents

Microtec Research MCC68K ANSI C Compiler Release 4.x Migration Manual

Introduction

This document describes the differences between the MCC68K 3.x compiler and the MCC68K 4.x (ANSI) compiler. This document should be used as a supplement to the compiler manuals and release notes.

Unix Installation Directory

The 3.x compiler was installed under the /mri directory. The 4.x compiler is installed under the /usr/mri directory.

By default, the compiler will look for standard include files in /usr/mri/include/mcc68k. The environment variable MRI_68K_INC may be used to specify an alternate list of standard include file directories. (For compatibility with the 3.x compiler, the environment variable C68K_INCLUDE is also supported.) See the Installation Guide for details.

By default, the driver will look for tool kit executables, libraries, and the default linker command file in the directories under /usr/mri. See the section *Compilation Driver Program* for the environment variables that can be used to override the defaults.

PC Installation Directory

The 3.x compiler package was installed under the following directories:

Directories	Compiler Package
\ASM68K	assembler, linker, librarian executables
\MCC68K	compiler executables, support files, include files
\MCC68K\68000	68000 run-time libraries
\MCC68K\68020Q	68020 quad-aligned run-time libraries
\MCC68K\68881Q	68020/68881 quad-aligned run-time libraries
\XRAY68K	debugger executable, support files

The 4.x compiler package is installed under the following directories:

Directories	Compiler Package
\ASM68K	assembler, linker, librarian executables
\MCC68K	compiler executables, support files, include files
\MCC68K\68000	68000 run-time libraries
\MCC68K\68020	68020 run-time libraries
\XRAY68K	debugger executable, support files

By default, the compiler will look for standard include files in \MCC68K. The environment variable MRI_68K_INC may be used to specify an alternate list of standard include file directories. See the Installation Guide for details.

By default, the driver will look for tool kit executables, libraries, and the default linker command file in the directories under \MCC68K and \ASM68K. See the section *Compilation Driver Program* for the environment variables that can be used to override the defaults.

VMS Installation Directory

The 4.x compiler files are installed in the same directories as the 3.x files were. Please refer to the Installation Guide for more details.

Compilation Driver Program (UNIX and PC)

The compiler is distributed with a UNIX-style driver program that can conveniently invoke the MCC68K C compiler, ASM68K assembler, and LNK68K linker.

The driver compiles, assembles, and links files to produce an executable file. Multiple file names may be given on the command line. Each file will be translated appropriately (based on its suffix), and all resulting object files will be linked together to produce an executable file.

A file's type is determined by its suffix:

Suffix	File's Type
.c	C source file
.i	C source file (output of preprocessor)
.s	assembly source file
.src	assembly source file
.o	object file
.obj	object file
.lib	library

The -S and -c options can be used to force the driver to stop after producing an assembly source file, or a relocatable object file. The -Vd option will cause the driver to display the commands it uses to invoke each of the tools. Please see the User's Guide and the *Compiler Options* section of this document for information on the driver options.

When the driver invokes the linker, it passes the linker the appropriate run-time library and the default linker command file (**mcc68k.cmd**). The driver selects which library to pass to the linker based on the -p and -Md options. It is very important that the same settings for these options are used on all driver invocations to produce a given executable. The -e option may be used to specify an alternate linker command file. This option also prevents the driver from passing the default library and default command file to the linker.

The driver and compiler have default directories they will use to find tool kit executables, run-time libraries, standard include files, and to write temp files. These default directories may be overridden by the following environment variables:

	UNIX Default	PC Default	Environ Var
executables	/usr/mri/bin	\MCC68K \ASM68K	MRI_68K_BIN
default .cmd file and libraries	/usr/mri/lib	\MCC68K	MRI_68K_LIB
std inc files	/usr/mri/include/mcc68k	\MCC68K	MRI_68K_INC
temp files	/tmp	\MCC68K\TMP	MRI_68K_TMP (if defined); otherwise, TMP

On the PC host, the driver will look for 68000 libraries in **\MCC68K\68000**, and the 68020 libraries in **\MCC68K\68020**. The environment variable **MRI_68K_LIB** may be used to specify an alternate base directory for the libraries. For example, if **MRI_68K_LIB** is set to **\MRI68K\LIB**, the driver will look for the 68000 libraries in **\MRI68K\LIB\68000**, and the 68020 libraries in **\MRI68K\LIB\68020**.

Compiler Options

The 4.x compiler options on UNIX and PC are closely modeled after the UNIX 'cc' compiler options. The 4.x compiler options on VMS follow VMS-style option syntax. The following sections show the mapping of the 3.x UNIX, PC, and VMS options to the 4.x options. The mapping tables refer to additional notes which can be found in the section *Notes*.

The mapping tables are not a complete list of the options available in the 4.x compiler. Please refer to the User's Guide and release notes for a full discussion of the 4.x options.

UNIX Options Mapping

This table shows the mapping of the UNIX options in the 3.x compiler to those in the 4.x compiler.

3.x Opt.		See Notes	4.x Opt.
-a	Mixed assembly language/C in .s file		-Fsm
-A	Perform all optimizations		-O
-b	*Split I/D space code*		
-B	Stablemem option		-Ob
-c	No padding in structs	3	(packed)
-C	Force preprocessor to preserve comments		(same)
-d	Output debug info for xray	5	-g
-D	Unix standard define		(same)
-e	Don't generate CLR instruction Allocate enums as small as possible	4 3,4	-nKc (packed)
-E	Debug trace option, generates FP info	8	-g -Kf
-f	noinit; generates storage but no zeros in bss	11	-Xp
-F	Use full path names for debug		-Gf
-g	Preassemble 68020/68881 instructions		
-G	Debug info on reg vars (debug=alive)	5	-g
-h	HP 64000 support		(same)
-i	Treat all routines as interrupt routines	3	(interrupt)
-I	Set search path for "" delimited includes		(same)
-j	Like -I, but for <> delimited includes		-J
-J	debug=lines; not required by xray		-Gl
-k	Force inits to code section	1	-aic

-K	Stack check gen		
-l file	Generate a list file with src/error		-l (to stdout)
-L	Disable label optimization		
-M	Strip user comments from assembly file		
-n	Suppress leading dot	2	-us
-N	Strip #include source from output assembly		-nFli (default)
-o file	Set output file name		(same)
-O	Perform no optimizations		
-p	Preprocess only to stdout		-E
-P	Turn off peephole optimizer		
-Q	Turn off stack optimizer		-nOc
-r	Place initialized data in separate section	1	(default)
-R	Turn off register allocator optimization		-nOR
-s	Treat bitfields as signed	10	
-S	Place character strings in a separate section	1	(default)
-t	Single precision floating point	9	-D_MATHLIBF
-T	Perform time optimization		-Ot (default)
-u	Map all names to upper case		
-U	Unix standard undefine		(same)
-V*n*	Force A*n* register to be reserved		-Mr*n*
-w	Suppress warnings		-Qw
-W	Generate position independent code		-Mcp
-X	Promote returned values to 32 bits	6	
-y	Syntax check only, no code gen		(same)
-Y	Generate position independent data		-Mdp
-Z*n*	Generate A*n* reg relative data		-Md*n*
-1	Generate 68010 code		-p68010
-2	Generate 68020 code	7	-p68020
-3	Generate 68020 code with quad-aligned data	7	-p68020
-4	Allow 32-bit displacements on position independent code		-Ml
-5	Allocate strings to code	1	-asc (default)
-6	Prepend an underscore, not a dot	2	-upu (default)
-7	Prolog/Epilog trace		-Kt
-8	Generate 68020/68881 code	7	-p68020 -f
-9	Generate 68020/68881 code with quad-aligned data	7	-p68020 -f

PC Options Mapping

This table shows the mapping of the PC options in the 3.x compiler to those in the 4.x compiler.

3.x Opt.		See Notes	4.x Opt.
call=noextend	Do not promote returned values to 32 bits	6	
call=stack	Stack check gen		
code=abs	Generate position dependent code		-Mca (default)
code=pc	Generate position independent code		-Mcp
cpu=68000	Generate 68000 code		-p68000 (default)
cpu=68010	Generate 68010 code		-p68010
cpu=68020	Generate 68020 code	7	-p68020
cpu=68020q	Generate 68020 code with quad-aligned data	7	-p68020
cpu=68881	Generate 68020/68881 code	7	-p68020 -f
cpu=68881q	Generate 68020/68881 code with quad-aligned data	7	-p68020 -f
data=abs	Generate position dependent data		-Mda (default)
data=A*n*	Generate A*n* reg relative data		-Md*n*
data=pc	Generate position independent data		-Mdp
debug	Output debug info for xray	5	-g
debug=alive	Debug info on reg vars (debug=alive)	5	-g
debug=filename	Use full path names for debug		-Gf
debug=lines	debug=lines; not required by xray		-Gl
debug=trace	Debug trace option, generates FP info	8	-g -Kf
define	Unix standard define		-D
hp	HP 64000 support		-h
id	Split I/D space code		
initvarintext	Force inits to code section	1	-aic
interrupt	Treat all routines as interrupt routines	3	(interrupt)
ipath	Set search path for "" delimited include		-I
(*) list	Generate a list file with src/error		-l (to stdout)
long	Allow 32-bit displacements on position independent code		-Ml
mix	Mixed assembly language/C in .src file		-Fsm
mix=nocomments	Strip user comments from assembly file		
mix=noinclude	Strip #include source from output assembly		-nFli (default)
noalign	No padding in structs	3	(packed)

noclr	Don't generate CLR instruction	4	-nKc
	Allocate enums as small as possible	3,4	(packed)
noinit	Generates storage but no zeros in bss	11	-Xp
nolp	Suppress leading dot	2	-us
nowarn	Suppress warnings		-Qw
opt=all	Perform all optimizations		-O
opt=nolabel	Disable label optimization		
opt=none	Perform no optimizations		
opt=nopeep	Turn off peephole optimizer		
opt=nopop	Turn off stack optimizer		-nOc
opt=noreg	Turn off register allocator optimization		-nOR
opt=stablemem	Stablemem option		-Ob
opt=time	Perform time optimization		-Ot (default)
(*) output	Set output file name		-o file
pp	Preprocess only to stdout		-E
pp=comment	Force preprocessor to preserve comments		-C
preassem	Preassemble 68020/68881 instructions		
reserve=A*n*	Force A*n* register to be reserved		-Mr*n*
rom	Place initialized data in separate section	1	(default)
rom=string	Place char strings in a separate section	1	(default)
signedbit	Treat bitfields as signed	10	
single	Single precision floating point	9	-D_MATHLIBF
spath	Like ipath, but for <> delimited includes		-J
strintext	Allocate strings to code	1	-asc (default)
syntax	Syntax check only, no code gen		-y
trace	Prolog/Epilog trace		-Kt
uc	Map all names to upper case		
undefine	Unix standard undefine		-U
underscore	Prepend an underscore, not a dot	2	-upu (default)

The options marked with a * were implemented in the 3.x compiler as positional parameters. The 3.x compiler on the PC had the following style command line:

`mcc68k` *source_filename, code_filename, list_filename*

The positional parameters *code_filename* and *list_filename* have been replaced by the -o and -l options.

VMS Options Mapping

This table shows the mapping of the VMS options in the 3.x compiler to those in the 4.x compiler (def means the option is on by default, + means the 4.x option is the same as the 3.x option.)

3.x Opt.		See Notes	4.x Opt.
[no]align	Controls padding in structs	3	(packed)
call=[no]extend	Do/Don't promote returned values to 32 bits	6	
call=[no]stack	Do/Don't generate stack check		
[no]clr	Do/Don't generate CLR instruction	4	+
	Allocate enums as small as possible	3,4	(packed)
code=abs	Generate position dependent code		+ (def)
code=pc	Generate position independent code		+
cpu=68000	Generate 68000 code		+ (def)
cpu=68010	Generate 68010 code		+
cpu=68020	Generate 68020 code	7	cpu=68020
cpu=68020q	Generate 68020 code with quad-aligned data	7	cpu=68020
cpu=68881	Generate 68020/68881 code	7	cpu=68020,fpu
cpu=68881q	Generate 68020/68881 code with quad-aligned data	7	cpu=68020,fpu
data=abs	Generate position dependent data		+ (def)
data=A*n*	Generate A*n* reg relative data		+
data=pc	Generate position independent data		+
[no]debug	Do/Don't output debug info for xray	5	+
debug=alive	Debug info on reg vars (debug=alive)	5	
debug=filename	Use full path names for debug		debug= fullfilename
debug=lines	Debug=lines; not required by xray		+
debug=[no]trace	Debug trace option, generates FP info	8	debug,frames
define	Unix standard define		+
[no]hp	Do/Don't generate HP 64000 support		+
[no]id	Split I/D space code		
[no]init	Do/Don't initialized uninits to zeros	11	weak=public weak=initzero weak= common(def)
initvarintext	Allocate init vars with code	1	init=code
noinitvarintext	Allocate init vars with data	1	init=data (def)
interrupt	Treat all routines as interrupt routines	3	
ipath	Set search path for "" delimited includes		+

list	Generate a list file with src/error		+
long	Use 32-bit displacements on position independent code		addr=long
nolong	Use 16-bit displacements on position independent code		addr=short
lp	Use leading dot	2	prepend=dot
nolp	Suppress leading dot	2	noprepend
mix	Show C source in .src file		show=source
nomix	Do not show C source in .src file		show= nosource (def)
mix=nocomments	Strip user comments from assembly file		
mix=noinclude	Strip #include source from output assembly		+ (def)
opt=all	Perform all optimizations		opt=all
opt=[no]label	Turn on/off label optimization		
opt=none	Perform no optimizations		
opt=[no]peep	Turn on/off peephole optimizer		
opt=[no] pop	Turn on/off stack optimizer		opt= [no]combine_pops
opt=[no]reg	Turn on/off register allocator optimization		+
opt=[no]stable	Stablemem option		+
opt=time	Perform time optimization		+ (def)
output=*name*	Set output file name		asm=*name*
pp	Preprocess only		+
pp=[no]comment	Force preprocessor to preserve comments		+
[no]preassem	Do/Don't preassemble 68020/68881 instructions		
reserve=A*n*	Force A*n* register to be reserved		+
rom	Place initialized data in separate section	1	(default)
norom	Place initialized data with uninit data	1	
rom=string	Place char strings in a separate section	1	(default)
[no]signedbit	Treat bitfields as signed	10	
single	Single precision floating point	9	define= _MATHLIBF
spath	Like ipath, but for <> delimited includes		+
stringsintext	Allocate strings with code	1	str=code (def)
nostringsintext	Allocate strings with data	1	str=data
[no]syntax	Syntax check only, no code gen		+
[no]trace	Prolog/Epilog trace		+

[no]uc	Do/Don't map all names to upper case		
undefine	Unix standard undefine		+
underscore	Prepend an underscore, not a dot	3	prepend= underscore(def)
nounderscore	Do not prepend underscore	3	noprepend
warn	Suppress warnings		supp=warnings
nowarn	Don't suppress warnings		nosuppress

Notes

1 See the *Section Names* section of this document.

2 See the *Naming Convention* section of this document.

3 See the *Packed and Interrupt Keywords* section of this document.

4 The 3.x noclr (-e) option had two disjoint effects. It has been split into the noclr (-nKc) option and the packed keyword for enums.

5 The 3.x debug=alive (-G) option was a superset of the 3.x debug (-d) option. Both options have been mapped to the 4.x debug (-g) option.

6 By default, the 3.x compiler would promote char/short return values to int. The 4.x compiler does not do this. If a function has a return type smaller than int, only the low order byte(s) are guaranteed to be valid. This may cause a problem if an extern function's return type is incorrectly declared:

```
extern int char_f1();  /* ERROR, actually returns char */
extern char char_f2(); /* correct */

i = char_f1();         /* ERROR, only low byte of
                          return value will be valid */
i = char_f2();         /* correct, will extend char
                          return value to int */
```

7 The 3.x compiler had options for generating 68020 code with even-aligned data (cpu=68020 (-2) or cpu=68881 (-8)) or quad-aligned data (cpu=68020q (-3) or cpu=68881q (-9)). The 4.x compiler always generates quad-aligned data with 68020 code. On the 68020, it is more efficient to access 4*n-sized data items on quad-word boundaries. The packed keyword may be used to override quad-aligning within structures.

8 The 3.x option debug=trace (-E) forced the compiler to generate a local stack frame (LINK/UNLK) for every function. The debug=trace option was implied by the debug option. To get the same functionality in the 4.x compiler use the -g and -Kf options or the debug and frames options on VMS.

9 The 3.x compiler provided single precision software floating point operations with the single (-t) option.This option:

1. Caused all operations involving floats to be done in single precision
2. Caused float parameters not to be promoted to double
3. Caused calls to functions defined in math.h to be re-directed to single precision versions of these functions.

In the 4.x compiler:

1. Operations involving floats are always done in single precision, if the ansi (-A) option is on (defaults on)
2. If prototypes are used, float parameters will not be promoted to double
3. To get the single precision versions of the math.h functions, the preprocessor symbol _MATHLIBF may be defined on the command line:

```
-D_MATHLIBF or /define=_MATHLIBF
```

If _MATHLIBF is defined the standard include file math.h will include mathf.h. mathf.h declares the standard math.h functions as functions which accept float arguments and return float values.

10 The 3.x option signedbit (-s) is no longer needed. Bitfields may be declared signed or unsigned.

11 The 3.x option noinit (-f) controlled whether or not uninitialized global variables were initialized to 0 when their space was allocated. A "weak external" is a global-level variable that is not initialized in its declaration. With the 4.x compiler, there are three ways to handle these variables.

With the weak=common (-Xc) option, weak externals will be output as XCOMs. If the variable is not externally defined in another module using XDEF, the linker will allocate it in the zerovars section. This option is the default.

With the weak=public (-Xp) option, weak externals will be output as XDEFs with the storage defined but not initialized. They will be placed by the compiler in the zerovars section. The zerovars section is set to zero at program start-up time.

With the weak=initzero (-X0) option, weak externals will be output as XDEFs and initialized to zero. They will be placed in the vars section.

Section Names

The 3.x compiler allocated code and data in two or three sections, based on the options initvarintext (-k), rom (-r), rom=stringonly (-S), and stringsintext (-5). The 4.x compiler and linker allocate code and data in the following sections:

Contents	Section Name	Addressed As
code	code	code (ROM)
uninit data	zerovars	data (RAM)
init data	vars	data (RAM)
const vars	const	code (ROM)
strings	strings	code (ROM)
compiler literals	literals	code (ROM)

Each of these sections may be independently located at link-time. If position independent code and data is to be used, the compiler must be told how each section should be addressed: as code (per the code= (-Mc) option), or as data (per the data= (-Md) option). The chart above shows the default address mode for each of the sections. Refer to the options in *Addressing Mode For Sections* in the User's Guide for information on specifying alternate addressing modes.

The following chart shows how the 3.x section names map to the 4.x section names:

3.x Name	4.x Name(s)
9	code
14	zerovars, vars
13	const, strings, literals
15	heap

In the 3.x compiler, the preprocessor symbols $CODESEG, $DATASEG, and $INITSEG were used to specify alternate section names. Refer to the options in *Name the Sections* in the User's Guide for information on specifying alternate section names in 4.x.

Refer to the *Default Linker Command File* section for changes that should be made to linker command files developed for the 3.x compiler.

Naming Convention

By default, the 3.x compiler would prepend a dot to all symbol names. The options nolp (-n) and underscore (-6) could be used to modify this behavior. The 4.x compiler defaults to prepending an underscore to all symbol names. Refer to the options in *Modify Naming Conventions* in the User's Guide for information on altering the naming convention.

Preprocessor Symbols

The following preprocessor symbols were predefined in the 3.x compiler:

```
mcc68k* MCC68K* mri* MRI*
mc68000 MC68000 m68k
```

These symbols are also predefined in the 4.x compiler. The symbols marked with a * are provided for backwards compatibility and may be removed in the future. It is recommended that uses of these symbols be changed to:

```
_MCC68K _MRI
```

In the 3.x compiler, the preprocessor symbols $INTERRUPT, $CODESEG, $DATASEG, and $INITSEG were used to declare interrupt handlers and to specify alternate section names. The 4.x compiler will issue errors for these symbols ($ is not a legal character in a preprocessor symbol). See the *Packed and Interrupt Keywords* section for information on declaring an interrupt handler. Refer to the options in *Name the Sections* in the User's Guide for information on specifying alternate section names.

For more information on these and other predefined symbols, refer to *The Preprocessor* chapter of the MCC68K reference manual.

Packed and Interrupt Keywords

The packed keyword may be used on structure and enumerated type definitions. The unpacked keyword is also supported, and is the default if neither is specified.

A packed structure will have no padding between structure members. In 68000/010 mode there may be one byte of padding at the end of a packed structure, to make the structure even-sized. In 68020 mode, there will be no padding in a packed structure.

A packed enumerated type will have, as a base type, the smallest integral type that can represent its values (char, short, int). The underlying integral type will be unsigned if there is at least an unsigned enumerated constant (e.g. 0u, 0xffffffff) in the enumeration definition, signed otherwise.

The interrupt keyword may be used to declare a function as an interrupt handler. Interrupt functions should have no parameters and should return the void type. For example:

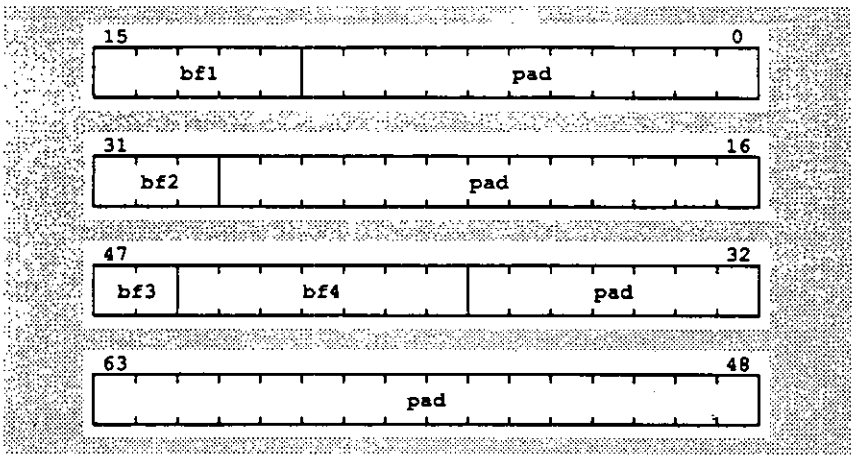
```
interrupt void handler(void)
{
    ...
}
```

See the Reference Manual for more information on declaring interrupt handlers.

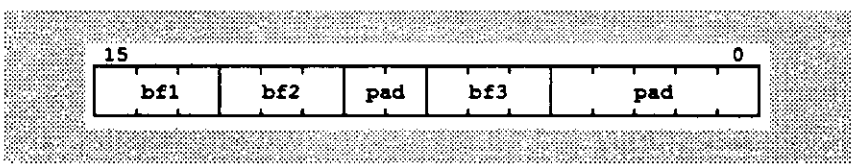
Bit Field Types

In addition to int and unsigned int bit fields, the 4.x compiler also supports char, short, and long (signed and unsigned) bit fields. Some examples:

```
struct {
    char bf1 : 5;
                        /* changing size, pad to next word */
    short bf2 : 3;
                        /* changing type, pad to next longword */
    int bf3 : 2;
    unsigned int bf4 : 7;
} s1;
```




```
struct {
    char bf1 : 3;
    unsigned char bf2 : 3;  /* changed sign */
    char bf3 : 3;           /* won't fit in current byte */
} s2;
```



As shown above, changing the type in a sequence of bit field members forces the next bit field to be aligned to its base-type boundary. Changing the sign in a sequence of bit field members does not force any alignment of the next bit field.

In-Line Assembly

The 3.x compiler implemented in-line assembly with the asm() statement. The 4.x compiler supports in-line assembly with the asm() pseudo-function. The asm() pseudo-function may also be specified as ASM().

The 4.x compiler may generate very different code than the 3.x compiler. All uses of in-line assembly should be examined in the context of the code generated by the 4.x compiler.

The 4.x implementation of in-line assembly is much more powerful than the 3.x implementation. Please see the MCC68K Reference Manual for more information.

The 4.x compiler also supports mixing C and assembly language with #pragma asm. For more information, refer to the MCC68K Reference Manual

Run-Time Libraries

The 3.x compiler was distributed with nine run-time libraries. The 4.x compiler is distributed with the following six run-time libraries:

	/cpu=68000 (-p68000)	/cpu=68020/addr=long (-p68020 -Ml)
/code=ab (-Mca) /data=ab (-Mda)	mcc68kab.lib	mcc68kab020.lib
/code=pc (-Mcp) /data=pc (-Mdp)	mcc68kpc.lib	mcc68kpc020.lib
/code=pc (-Mcp) /data=a5 (-Md5)	mcc68ka5.lib	mcc68ka5020.lib

For UNIX and VMS hosts, the following chart shows which 4.x library you should use, based on the 3.x library you were using:

3.x Library	4.x Library
mcc68kab.lib	mcc68kab.lib
mcc68kab020q.lib	mcc68kab020.lib
mcc68kab881q.lib	mcc68kab020.lib
mcc68kpc.lib	mcc68kpc.lib
mcc68kpc020q.lib	mcc68kpc020.lib
mcc68kpc881q.lib	mcc68kpc020.lib
mcc68ka5.lib	mcc68ka5.lib
mcc68ka5020q.lib	mcc68ka5020.lib
mcc68ka5881q.lib	mcc68ka5020.lib

For the PC host, the following chart shows which 4.x library you should use, based on the 3.x library you were using:

3.x Library	4.x Library
\mcc68k\68000\mcc68kab.lib	\mcc68k\68000\mcc68kab.lib
\mcc68k\68020q\mcc68kab.lib	\mcc68k\68020\mcc68kab.lib
\mcc68k\68881q\mcc68kab.lib	\mcc68k\68020\mcc68kab.lib
\mcc68k\68000\mcc68kpc.lib	\mcc68k\68000\mcc68kpc.lib
\mcc68k\68020q\mcc68kpc.lib	\mcc68k\68020\mcc68kpc.lib
\mcc68k\68881q\mcc68kpc.lib	\mcc68k\68020\mcc68kpc.lib
\mcc68k\68000\mcc68ka5.lib	\mcc68k\68000\mcc68ka5.lib
\mcc68k\68020q\mcc68ka5.lib	\mcc68k\68020\mcc68ka5.lib
\mcc68k\68881q\mcc68ka5.lib	\mcc68k\68020\mcc68ka5.lib

Support files are included in the 4.x libraries. If you do not need to modify these files, it is no longer necessary to explicitly load them in the linker command file. If you do need to modify the support files (they are also distributed in source form), you should explicitly load your modified version in the linker command file.

Linking 4.x Objects With 3.x Libraries

Some users may wish to compile with the 4.x compiler, but continue to link with libraries built with the 3.x compiler. The following differences between 4.x and 3.x effect combining 3.x libraries with 4.x objects:

1. The 4.x compiler uses a different naming convention than the 3.x compiler. 3.x used leading dot. 4.x uses leading underscores. Libraries built with 3.x will contain symbols with leading dots. The 4.x **prepend=dot (-upd)** option can be used to make the 4.x compiler use leading dots rather than underscores.
2. The 4.x compiler uses different intrinsic functions than 3.x. The compiler will generate calls to intrinsic functions for:

   ```
   1. 'float'/'double' operations if no 68881
   2. 'long' operations if no 68020
   ```

   The 4.x intrinsics are, of course, not in the 3.x libraries, and must be loaded from the 4.x libraries. (The **prepend=** option does not effect the naming convention used for the intrinsic routines.)

3. The 4.x compiler uses different section names than 3.x:

3.x Sections	4.x Sections
9	code
14	zerovars, vars
13	const, strings, literals
15	heap

The LNK68K alias command can be used to map the 3.x section names to the 4.x section names.

Instructions for linking 4.x objects with the 3.x library:

1. Compile with the 4.x prepend=dot (-upd) option to use a leading dot (to match the 3.x naming convention) and the spath (-J) option (to select the 3.x standard include files).
2. Make a copy of the 4.x sample linker command file (on UNIX and PC: mcc68k.cmd, on VMS: sieve.opt), and modify it as follows:
   1. Comment-out the extern command used to force loading of the initialization routine from the 4.x library:

```
* extern ENTRY
```

   2. Add the following alias commands to map the 3.x section names to the 4.x section names:

```
alias code,9
alias zerovars,14
alias vars,13
alias heap,15
```

   3. Add the following load commands immediately before the end command:

```
load entry.o                    ; 3.x initialization routine
load sbrk.o                     ; 3.x support routine
load csys68k.o                  ; 3.x support routine
load inchrw.o                   ; 3.x support routine
load outchr.o                   ; 3.x support routine
load /usr/mri/lib/mcc68kab.lib ; 4.x library
                                ; (pull intrinsics only)
load /mri/lib/mcc68kab.lib      ; 3.x library
```

(VMS and PC users: change file names and directory names appropriately.)

4. Invoke the linker with your modified linker command file (mcc68k_3x.cmd or sieve_3x.opt):

   UNIX and PC:

   ```
   lnk68k -o test.x -m -c mcc68k_3x.cmd test.o > test.map
   ```

   VMS:

   ```
   l68k sieve_3x.opt/opt
   ```

Default Linker Command File (UNIX and PC)

The 3.x compiler was distributed with a sample linker command file, sieve.cmd. Users were expected to create their own linker command files, based on sieve.cmd.

The 4.x compiler on UNIX and PC is distributed with a default linker command file, mcc68k.cmd. As described in the *Compilation Driver Program* section, the driver invokes the linker with:

1. the objects produced by compiling and assembling
2. the run-time library
3. and the default linker command file.

Users may modify the default linker command file to suit their needs, or use the Pass Command File to Linker option (-e) to specify an alternate linker command file. If the Pass Command File to Linker option is used, the alternate command file should be based on the default command file, and should contain an explicit load of the appropriate run-time library.

The default linker command file:

```
;
;           MCC68K V4 Default Linker Command File
;
listabs     publics,internals
listmap     publics
format      ieee
extern      ENTRY                   ; force load of initialization routine
;
; If linking with MCC68K 3.x objs, enable the next 4 lines:
;
;   alias code,9          ;  9 was default value of $CODESEG in 3.x
;   alias zerovars,14     ; 14 was default value of $DATASEG in 3.x
;   alias vars,13         ; 13 was default value of $INITSEG in 3.x
;   alias heap,15         ; 15 was heap section in 3.x
;
sectsize    heap=$8000    ;set size of heap
sectsize    stack=$1000   ;set size of stack
common      stack=$f000   ;set starting address of stack section
```

```
;
order       stack                                   ; stack section
order       literals,strings,const,initfini,code    ; ROM sections
order       ??INITDATA                      ; ROM section for init values
order       vars,zerovars,tags,ioports,heap         ; RAM sections
;
; If using A5-relative data addressing, enable the next 2 lines:
;
;index ?a5,vars,$8000  ;A5 reg must be set to ?A5 in entry.s
;load  /usr/mri/lib/mcc68ka5XX.lib ; Modify to access correct library

;
; If using run-time initialization of RAM, enable the next 3 lines (also
; need to recompile csys.c using the command line switch "-D_INITDATA".):
;
;initdata  vars           ;put init values in ??INITDATA
;load csys.o              ; modify to call __initcopy
;load /usr/mri/lib/mcc68kXX.lib  ; Modify to access correct library
end
```

Incompatibilities

The 4.x compiler is (with minor exceptions) C-source-level compatible with the 3.x compiler. It is not assembly-source-level compatible with 3.x. All C source modules should be re-compiled and assembled with the 4.x compiler. The following incompatibilities exist:

1. In the 3.x compiler, return values of struct functions were returned through a static area. This technique was not re-entrant. In the 4.x compiler, return values of struct functions are returned through a hidden pointer. This technique is re-entrant.
2. The naming convention has changed. The 3.x compiler prepended a dot to all symbols. The 4.x compiler prepends an underscore to all symbols. The *Modify Naming Conventions* option can be used to force the compiler to prepend a dot rather than an underscore.
3. The number of sections and their names has changed. The 3.x compiler allocated code and data in two or three sections. The 4.x compiler and linker allocate code and data in six sections. The *Name The Sections* options and the LNK68K alias command may be used to map the 4.x section names to the 3.x section names:

   UNIX- and PC-hosted compiler options:

   ```
   -NT9 -NZ14 -NI14 -NC13 -NL13 -NS13
   ```

   VMS-hosted compiler options:

   ```
   /rename=(code=9,data=14,init=14,const=13,lit=13,str=13)
   ```

LNK68K command:

```
alias zerovars,14
```

As an alternative to specifying the Name The Sections options on the command line, the UNIX-flavor options may be specified in the C source file:

```
#pragma options -NT9 -NZ14 -NI14 -NC13 -NL13 -NS13
```

4. Any code which relies on particular code being generated by the compiler is suspect. For example, if an assembly language routine called by a C routine relies on the C routine having a LINK instruction, or relies on the C routine placing variables in certain registers, the assembly routine will behave unexpectedly.

   This also applies to user in-line assembly code. The 4.x compiler will often allocate variables in registers differently than the 3.x compiler. Refer to section *In-Line Assembly* for other differences.

5. The preprocessors symbols $INTERRUPT, $CODESEG, $DATASEG, and $INITSEG are no longer supported. The 4.x compiler will issue error messages for these symbols. The $INTERRUPT preprocessor symbol has been replaced by the interrupt keyword. The other preprocessor symbols have been replaced by the Name The Sections options. These changes may require C-source modifications.

6. The 3.x compiler and many other pre-ANSI compilers allowed "." in a preprocessor symbol, for example:

   ```
   #if MY.SYMBOL
   ```

   The ANSI standard specifies that preprocessor symbols follow the same rules as C identifiers "." not allowed). The 4.x compiler will issue an error if a preprocessor symbol is not a legal identifier. The preprocessor symbol should be modified (perhaps by replacing "." with "_").

7. 3.x compiler allowed spaces within assignment operators, for example:

   ```
   k + = 2;
   ```

   The 4.x compiler does not allow this. The statement should be specified as:

   ```
   k += 2;
   ```

8. The 3.x compiler would accept 3 ways of initializing an aggregate:

   1) Flat initialization {.....}
   2) Partially parenthesized; some of the {'s are there but not all of them.
   3) Fully parenthesized.

   The 4.x compiler, conforming to the ANSI standard, implements #1 and #3. Please consider the following example which shows #2 versus #3.

```
 1 struct tag
 2     {
 3     int ar[2][5];  /* ONE member: an array of array */
 4     } var=
 5 {
 6     { 1, 2, 3, 4, 5 },    /* initialize first member */
 7     { 1, 2, 3, 4, 5 }     /* initialize ?second? ?member? */
       ^
>> (E) too many initializers for "var"
 8 };
 9
10 struct tag
11     {
12     int ar[2][5];  /* ONE member */
13     } okvar=
14 {
15   {
16     { 1, 2, 3, 4, 5 },    /* initialize okvar.ar[0] */
17     { 1, 2, 3, 4, 5 }     /* initialize okvar.ar[1] */
18   }
19 };
```

Notes

Microtec Research, Incorporated

Microtec Research, Incorporated, Santa Clara, California, has been creating solutions for microprocessor developers since 1974. Founded by specialists in microprocessor technology, Microtec Research has provided over a decade of experience in the design, development, and support of software products for microprocessors.

Microtec Research's languages and software development tools have been designed for the professional developer of microprocessor applications. Microtec Research's thousands of customers throughout the world represent virtually every business, scientific, governmental, and educational activity requiring microprocessor program generation and testing.

Microtec Research Inc.

2350 Mission College Blvd.

Santa Clara, CA 95054

Tel. 408.980.1300

Toll Free 800.950.5554

FAX 408.982.8266

Printed in U.S.A.